AD-A120 376 SYSTEMS ARCHITECTS INC RANDOLPH MASS F/G 9/2
COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK. VOLUME I. INTROD--ETC(U)
MAY 82 F19628-80-C-0207
UNCLASSIFIED ESD-TR-82-143(1) NL

1 of 1

END
DATE
FILMED
11 82
DTIC

(12)

ESD-TR-82-143(I)

AD A120375

INTRODUCTION AND GENERAL INSTRUCTIONS FOR COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK. VOLUME I.

Systems Architects, Inc.
50 Thomas Patten Drive
Randolph, MA 02368

May 1982

Approved for public release;
Distribution unlimited.

Prepared for

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
DEPUTY FOR TECHNICAL OPERATIONS AND
PRODUCT ASSURANCE
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

Robert V. Vieraitis Jr.
ROBERT V. VIERAITIS, Jr., 1Lt, USAF
Project Officer

James W. Neely, Jr.
JAMES W. NEELY, Jr., Lt Col, USAF
Chief, Computer Engineering
Applications Division

FOR THE COMMANDER

Walter W. Turgiss
WALTER W. TURGISS
Acting Director, Engineering and Test
Deputy for Technical Operations
and Product Assurance

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-82-143(I)	2. GOVT ACCESSION NO. AD-A120375	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Introduction and General Instructions for Computer Systems Acquisition Metrics Handbook. Volume I.		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Systems Architect, Inc.		8. CONTRACT OR GRANT NUMBER(s) F19628-80-C-0207
9. PERFORMING ORGANIZATION NAME AND ADDRESS Systems Architects, Inc. 50 Thomas Patten Drive Randolph, MA 02368		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Electronic Systems Division (TOEE) Hanscom AFB Massachusetts 01731		12. REPORT DATE May 1982 13. NUMBER OF PAGES 56
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer systems
Metrics
Quality assurance
Software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This volume provides an overview to a standard set of procedures for qualitatively specifying and measuring the quality of a computer software system during its acquisition life cycle.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

Section	Title	Page
I	INTRODUCTION TO SOFTWARE METRICS	
	1.1 OVERVIEW	I-1
	1.2 SOFTWARE METRICS FRAMEWORK	I-1
	1.2.1 Quality Factors	I-3
	1.2.2 Criteria	I-3
	1.2.3 Metrics	I-9
	1.2.4 Data Elements	I-9
II	SOFTWARE DEVELOPMENT LIFE CYCLE MODEL	
	2.1 INTRODUCTION	II-1
	2.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE MODEL FOR METRICS	II-1
	2.3 THE SOFTWARE DEVELOPMENT LIFE CYCLE MODEL FROM ESD GUIDEBOOK SERIES	II-1
	2.4 SOFTWARE DEVELOPMENT PRODUCTS	II-3
	2.4.1 First Metric Application Information Requirements	II-4
	2.4.2 Update Information Requirements	II-8
	2.4.3 Documentation Considerations	II-12
	2.4.3.1 Contents of Documents	II-12
	2.4.3.2 Product Mapping	II-12
	2.4.3.3 Appropriate Levels of Applying Metrics	II-14
	2.4.3.4 Sequencing of Metric Application	II-16
	2.5 APPLICABILITY OF SOFTWARE METRICS THROUGH THE LIFE CYCLE	II-16
III	HANDBOOK FRAMEWORK	
	3.1 COMPONENTS OF COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK	III-1
	3.2 INTERRELATIONSHIP OF SOFTWARE METRIC'S AND HANDBOOK'S FRAMEWORKS	III-1
IV	HOW THE HANDBOOK WORKS	
	4.1 INTRODUCTION	IV-1
	4.2 STEPS FOR USING THE SOFTWARE METRICS HANDBOOK	IV-1
	4.3 IDENTIFICATION OF WORKSHEETS	IV-2
	4.4 DATA ELEMENT DICTIONARY (DED)	IV-4

PRECEDING PA[illegible]

TABLE OF CONTENTS (Cont'd)

Section	Title	Page
V	QUALITY FACTOR SELECTION	
	5.1 INTRODUCTION	V-1
	5.2 QUALITY FACTOR TRADE OFFS	V-1
	5.3 SIGNIFICANCE OF QUALITY FACTORS IN C^3 SYSTEMS	V-3
VI	QUALITY FACTOR EVALUATION	
	6.1 POST DATA COLLECTION	VI-1
	6.2 EVALUATION	VI-1

SECTION I

INTRODUCTION TO SOFTWARE METRICS

1.1 OVERVIEW

This Handbook contains a standard set of procedures to quantitatively specify and measure the quality of a computer software system during its acquisition life cycle. These quantitative measures, or metrics, provide the user with a tool to better assess the system's development and potential performance throughout the acquisition phases.

The metrics are calculated from the answers to questions, called data elements in this Handbook, which also serve as a checklist to aid Software Quality Assurance. These metrics are a tool for current Software Quality Assurance practices. They are an added feature to current tools and techniques utilized in Software Quality Assurance practices.

The Handbook is tailored specifically to address embedded Command Control and Communications (C^3) computer systems. Efforts to apply the procedures to other than C^3 systems may require reworking by the user of the materials contained in the Handbook.

1.2 SOFTWARE METRICS FRAMEWORK

Software Metrics are a set of measurements for measuring essential aspects of software systems. Realistic assessments and ratings are based on the measurements so that the overall quality of a system's software can be made visible.

There are four levels in the Software Metrics Framework presented in this Handbook: (1) Factors, (2) Criteria, (3) Metrics, and (4) Data Elements. This Framework is illustrated in Figure I-1.

Each level of the Framework is defined in the following subsections. Conflicting or different definitions outside of this Handbook are not applicable to Software Metrics, and conversely,

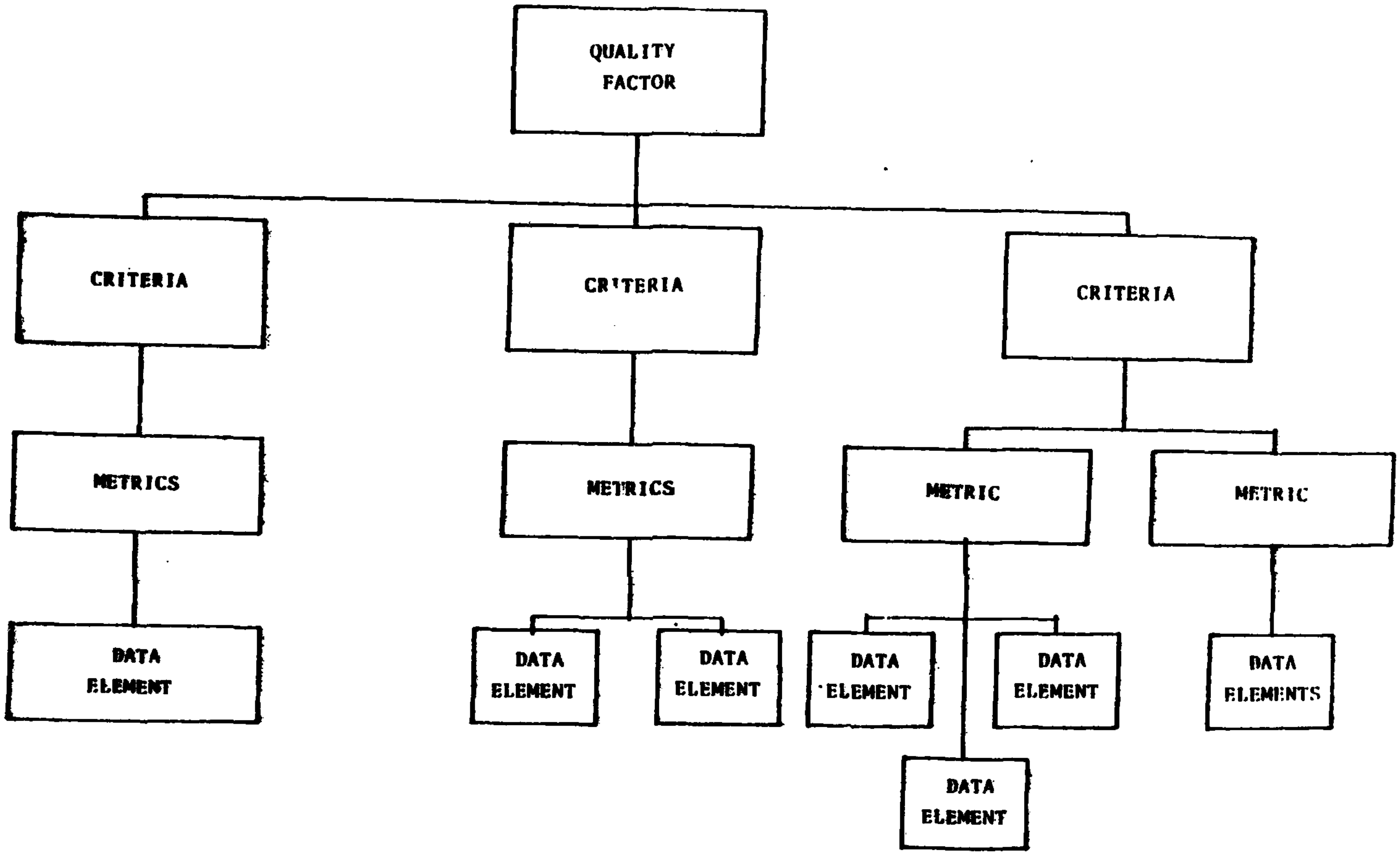


FIGURE I-1

SOFTWARE METRICS FRAMEWORK

the definitions found within this Handbook are not necessarily applicable to other concepts.

1.2.1 Quality Factors

Quality Factors are management-oriented terms which define desirable characteristics of the software systems under development from a management perspective. Presence of these Quality Factors improve the probability of producing the desired software system. A quantitative analysis of the Quality Factors will indicate areas of weakness and strength in the system. Maintainability and Integrity are two examples of Quality Factors. Maintainability refers to the effort required to locate and fix an error in an operational program. Integrity refers to the extent to which access to software or data by unauthorized persons can be controlled.

Eleven (11) Quality Factors have been selected for this Handbook. TABLE I-A contains a complete list of the Quality Factors and their definition. A discussion of selecting appropriate Quality Factors for a specific system is found in Section V, "Quality Factor Selection".

1.2.2 Criteria

Criteria form the next level of the Framework under the Quality Factors. Criteria are software-oriented terms that describe software attributes. Each Quality Factor has two or more Criteria related to it in a hierarchy. Consistency and Simplicity are two examples of Criteria related to the Quality Factor "Maintainability". Consistency and Simplicity are two examples of Criteria related to the Quality Factor "Maintainability". Consistency refers to those attributes of the software that provide implementation of function in the most understandable manner.

Twenty-two (22) Criteria have been included in this Handbook. TABLE I-B contains a complete list of the Criteria and a definition for each. Figure I-2 shows the relationship of the Criteria to the Quality Factors.

FACTOR	DEFINITION
Correctness (Co)	Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Reliability (Re)	Extent to which a program can be expected to perform its intended function with required precision.
Efficiency (Ef)	The amount of computing resources and code required by a program to perform a function.
Integrity (It)	Extent to which access to software or data by unauthorized persons can be controlled.
Usability (Us)	Effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability (Ma)	Effort required to locate and fix an error in an operational program.
Testability (Te)	Effort required to test a program to insure it performs its intended function.
Flexibility (Fx)	Effort required to modify an operational program.
Portability (Po)	Effort required to transfer a program from one hardware configuration and/or software system environment to another.
Reusability (Ru)	Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform.
Interoperability (Ip)	Effort required to couple one system with another.

TABLE I-A

QUALITY FACTOR TABLE

CRITERION	DEFINITION
Traceability	Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment.
Completeness	Those attributes of the software that provide full implementation of the functions required.
Consistency	Those attributes of the software that provide uniform design and implementation techniques and notation.
Accuracy	Those attributes of the software that provide the required precision in calculation and outputs.
Error Tolerance	Those attributes of the software that provide continuity of operation under nonnominal conditions.
Simplicity	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity).
Modularity	Those attributes of the software that provide a structure of highly independent modules.
Generality	Those attributes of the software that provide breadth to the function performed modules.
Expandability	Those attributes of the software that provide for expansion of data storage requirements or computational functions.
Instrumentation	Those attributes of the software that provide for the measurement of usage or identification of errors.
Self - Descriptiveness	Those attributes of the software that provide explanation of the implementation of a function.

TABLE I-B

CRITERIA TABLE

CRITERION	DEFINITION
Execution Efficiency	Those attributes of the software that provide for minimum processing time.
Storage Efficiency	Those attributes of the software that provide for minimum storage requirements during operation.
Access Control	Those attributes of the software that provide for control of the access of software and data.
Access Audit	Those attributes of the software that provide for an audit of the access of software and data.
Operability	Those attributes of the software that determine operation and procedures concerned with the operation of the software.
Training	Those attributes of the software that provide transition from current operation or initial familiarization.
Communicativeness	Those attributes of the software that provide useful inputs and outputs which can be assimilated.
Software System Independence	Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.).
Machine Independence	Those attributes of the software that determine its dependency on the hardware system.
Communications Commonality	Those attributes of the software that provide the use of standard protocols and interface routines.
Data Commonality	Those attributes of the software that provide the use of standard data representations.

TABLE I-B (Continued)

CRITERIA TABLE

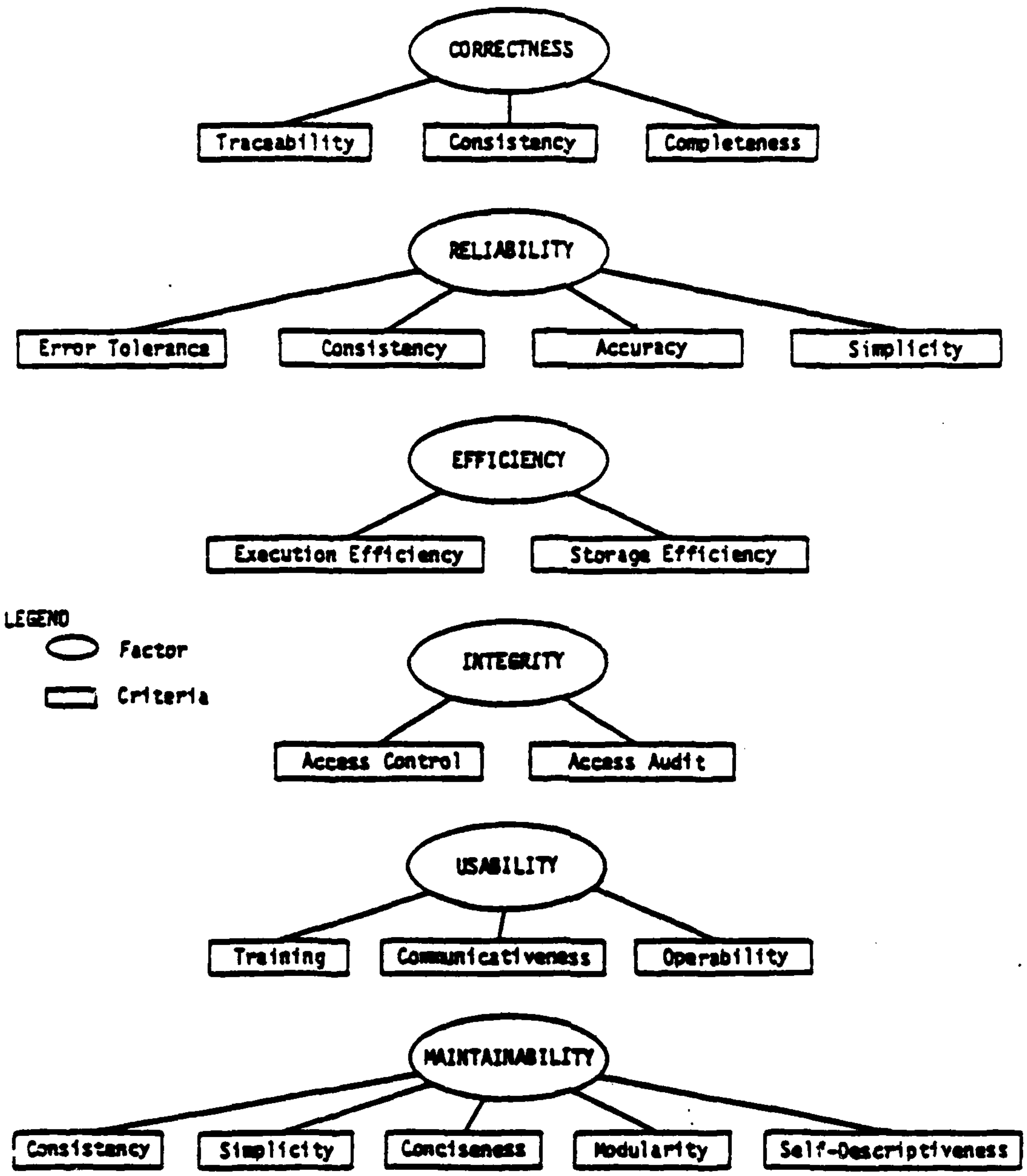


FIGURE I-2
RELATIONSHIP OF CRITERIA TO SOFTWARE QUALITY FACTORS

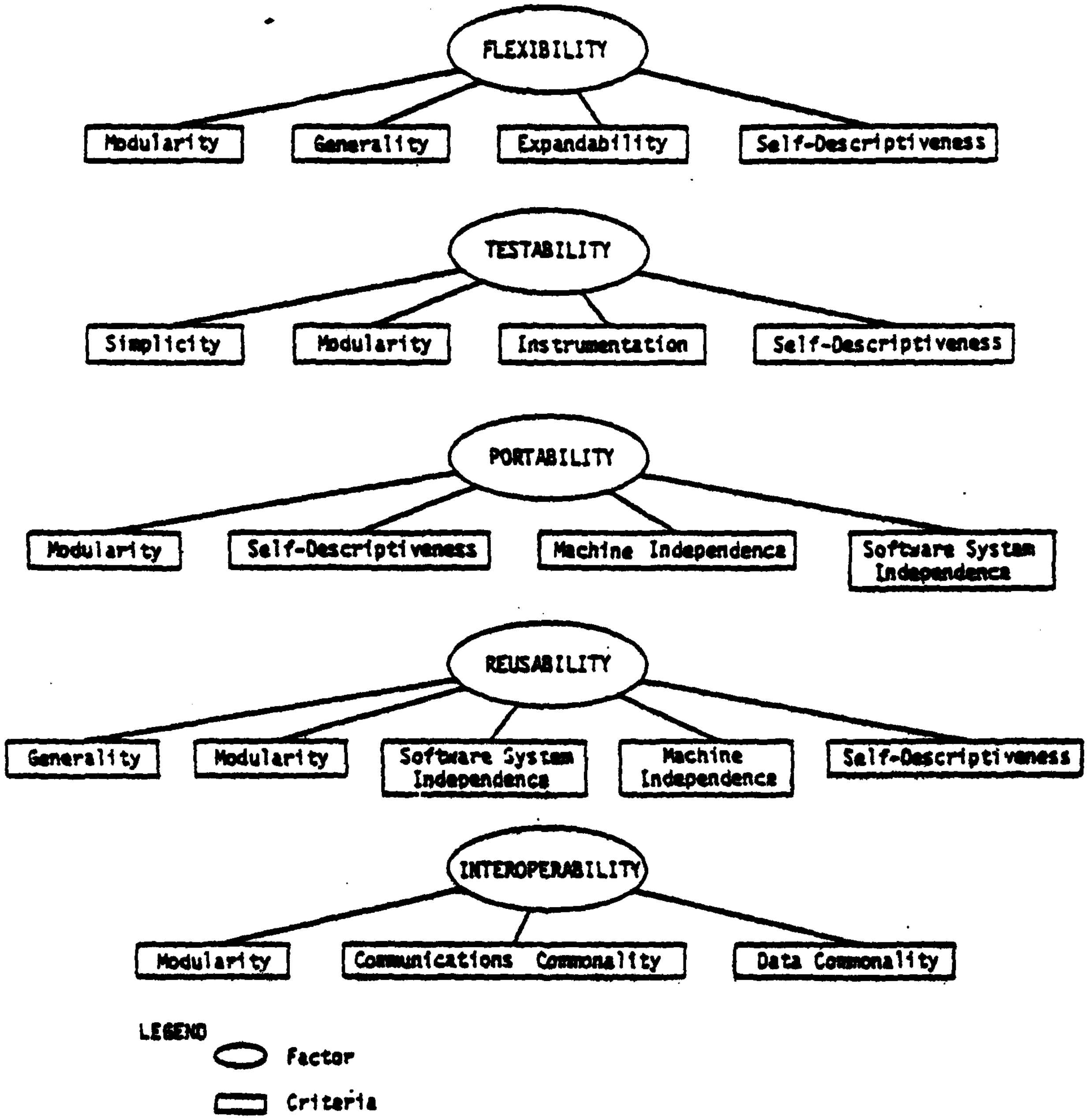


FIGURE I-2

RELATIONSHIP OF CRITERIA

TO SOFTWARE QUALITY FACTORS (Continued)

1.2.3 Metrics

Metrics are at the third level of the Framework and represent the measurable aspects of the Criteria. Each Criterion has at least one Metric, while some Criteria have several Metrics associated with them. Design Structure Measure, Structured Programming Check, and Complexity Measure are examples of the Metrics connected with the Criterion "Simplicity".

Thirty-seven (37) Metrics have been identified in this Handbook. TABLE I-C has a complete list of the Metrics.

1.2.4 Data Elements

Data Elements are at the fourth level of the Framework. They are quantifyable questions that combine to produce a Metric Value. An algorithm based on the answers to Data Elements for each Metric determines the value for each Metric. Data Elements are questions about the software and software development products that call for either YES-NO or numeric type responses. Each Metric has at least one Data Element. For example, Hierarchical Structure, Module Independence, and Size of Data Base are the Data Elements that comprise the Metric "Design Structure Measure". Hierarchical Structure asks the question: "Is a hierarchical chart provided which identifies all modules in the system?" Module Independence asks: "Is the module independent of the source of the input or the destination of the output?" And size of Data Base asks: "Number of unique data items in a data base?" All 126 Data Elements are defined in a Data Element Dictionary which is provided as part of this Handbook.

Access Audit Checklist
Access Control Checklist
Accuracy Checklist
Communications Commonality Checklist
User Input Interface Measure
User Output Interface Measure
Completeness Checklist
Procedure Consistency Measure
Data Consistency Measure
Data Commonality Checklist
Performance Requirements Check
Iterative Processing Efficiency Measure
Data Usage Efficiency Measure
Error Tolerance Control Checklist
Recovery From Improper Input Data Checklist
Recovery From Computational Failures Checklist
Recovery From Hardware Faults Checklist
Recovery From Device Errors Checklist
Data Storage Expansion Measure
Extensibility Measure
Implementation For Generality Checklist
Module Testing Measure
Integration Testing Measure
System Testing Measure
Machine Independence Measure
Modular Implementation Measure
Operability Checklist
Quantity of Comments
Effectiveness of Comments Measure
Descriptiveness of Implementation Language Measure
Storage Efficiency Measure
Design Structure Measure
Complexity Measure
Measure of Coding Simplicity
Software System Independence Measure
Training Checklist
Traceability Checklist

TABLE I-C

METRICS

SECTION II

SOFTWARE DEVELOPMENT LIFE CYCLE MODEL

2.1 INTRODUCTION

The Software Development Life Cycle Model is a management tool that describes the major activities of a software development effort in the order in which they are performed. There have been a number of Software Development Life Cycle Models developed, each for a particular environment. The Software Development Life Cycle Model adapted for use in relationship to the Metric is a simple one, generic in nature. The idea behind selecting such a simple model is that it can be mapped into more particular models so that the Metrics can be applied in a wide range of environments.

2.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE MODEL FOR METRICS

The Software Development Life Cycle Model for Metrics has five phases; (1) Requirements Analysis (2) Preliminary Design (3) Detail Design (4) Implementation and (5) Test and Integration. As of the writing of this Handbook,Metrics have been developed for the first four phases and are still under development for the Test and Integration phase and therefore Metrics for this phase are not included in this Handbook. We discuss this phase of the model because update information is collected during this phase.

2.3 THE SOFTWARE DEVELOPMENT LIFE CYCLE MODEL FROM ESD GUIDEBOOK SERIES

The Software Development Life Cycle Model from the ESD Guidebook Series has seven phases; (1) Conception (2) Analysis (3) Design (4) Coding and Checkout (5) Test and Integration (6) Installation and (7) Operation and Support. Figure II-1 maps the

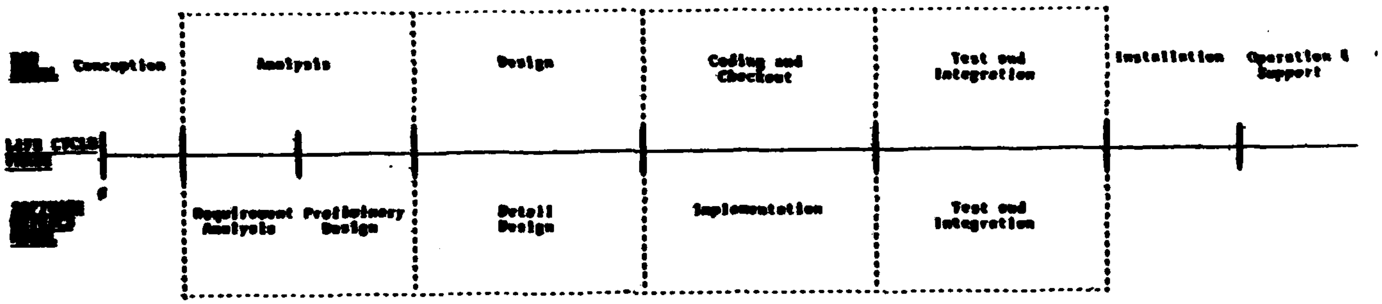


FIGURE II-1

COMPARISON OF ESD's AND SOFTWARE METRIC'S LIFE CYCLE MODELS

Metrics model into the Guidebook series model. As can be seen in Figure II-1 there are three major differences between the two models; (1) The Metrics model does not consider three of the Guidebook phases, "Conception", "Installation" or "Operation and Support" (2) The Metrics model's "Requirements Analysis" and "Preliminary Design" phases correspond to the Guidebook's "Analysis" phase and (3) The Metrics models "Detail Design" and "Implementation" correspond to the Guidebook's "Design" and "Coding and Checkout". This particular mapping is based on the description of these phases in the Guidebook and explains why Metrics "Preliminary Design" is mapped to "Analysis" and not to "Design". It should be made clear that these models are descriptions of the order in which events usually take place and are not "Directives" that demand events take place in that order. Therefore there are no "Hard and Fast" rules for any mapping, the users judegement must be exercised in applying the metrics using the life cycle models involved as a guide, not a directive.

2.4 SOFTWARE DEVELOPMENT PRODUCTS

In order to apply the Metrics to measure software quality, two types of information must be available. The first type of information required is the original organized documentation that describes the system being measured. The second type of information required by Software Metrics is the revisions to the original documents. The original documents are utilized by Software Metrics in the first application efforts, in order to catch development problems early in the life cycle. The revised documents are measured during the update process in order to determine whether these problems have been corrected adequately. This information is available in what can be generically called "Software Development Products". These "Software Development Products" are produced throughout the "Software Development Life Cycle" and include such products as; requirement specifications, design documentation, system/module diagrams and flowcharts with PDL, code listings and complete sets of plans and procedures. For a more detailed understanding, as an example use the "Products" tied to the ESD Guidebook Life-Cycle Events Guidebook. These are "Products" that are usually produced during the development and acquisition of an embedded software system. Table II-A lists these "Products" as they appear in the Guidebook and organized under the Guidebooks Software Development Life Cycle Model. By using the Table II-A in conjunction with Figure II-1 it is possible to map these "Products" into the Metric Software Development Model. The results of this mapping is demonstrated in Figure II-2 and in more detail in Figure II-3. Notice that products of later phases provide information required for updates of Metrics originally calculated in earlier phases.

2.4.1 First Metric Application Information Requirements

The initial types of information required by Software Metrics (for the Requirement Analysis Metrics) are Tradeoff Study Reports and System Specifications. These documents are usually generated during an analysis of system alternatives. Other documents necessary for Requirements Analysis phase Metrics are authenticated development specifications for each CPCI and any specification or ICD changes, which are generated during allocation of requirements to the computer program. FIGURE II-3 details the activities and products in the Requirements Analysis and the Preliminary Design phase.

A second set of information is required in order to perform Preliminary Design phase Software Metrics. The parts of the draft product specifications containing design approaches for each CPCI, which are generated during the requirements allocation process, together with the minutes and action item responses that are produced during a Preliminary Design Review are both required for the Software Metrics in this phase. FIGURE II-3 describes the activities and products needed for the Preliminary Design phase, as well as Requirement Analysis phase.

ANALYSIS PHASE

Activity	Product(s)
A. Devise & analyze alternatives for the system, Segment (if any), or any Software Subsystem directly containing the Computer Program.	A.1. Tradeoff study reports. 2. Initial or Authenticated System Specification & Segment Specifications (if any).
B. Allocate requirements to the Computer Program: i.e. Functions. Performance (e.g. response times). Interface (with others) Design constraints (e.g., prescribed algorithms, core & processing time budgets). Testing.	B.1. Authenticated Development Specification for each CPCI. 2. Possible higher-level specification, and ICD, changes 3. Parts of draft Product Specifications containing design approaches for each CPCI.
C. Conduct PDR(s) for the Computer Program's CPCI(s).	C. PDR minutes and action item responses.

DESIGN PHASE

Activity	Product(s)
A.1. Define algorithms not previously prescribed. 2. Design data storage structures. 3. Define Computer Program Logic.	A.1. Functional flowcharts. 2. Detailed flowcharts. 3. Data format descriptions. 4. Descriptions of algorithms not previously prescribed.
B. Allocate Computer Program requirements internally (e.g., to CPCs).	B Preliminary Product Specifications, including the above.
C. Test Planning.	C.1. System, Segment (if any) and CPCI Test Plans. 2. Prelminary CPCI Test Procedures.
D. CDR(s) for the Computer Program's CPCI(s).	D. CDR minutes & action item responses.

CODING AND CHECKOUT PHASE

Activity	Product(s)
A. Coding.	A-B Code.
B. Limited checkout of compiler or assembly units.	
C. Corresponding logic & data structure revisions.	C. Altered Product Specifications, including compiler/assembly listings.

TEST AND INTEGRATION PHASE

Activity	Product(s)
A. Test Planning.	A.1. Final CPCI Test Procedures. 2. Segment (if any) and system-level Test Procedures.
B. Module tests.	B-D1. Test Report. 2. Computer Program coding changes. 3. Modified Product Specifications. 4. Possible high-level specification, and ICD, changes.
C. CPCI test (PQT & FQT).	
D. Software Subsystem integration.	

TABLE II-A

LIFE CYCLE EVENTS

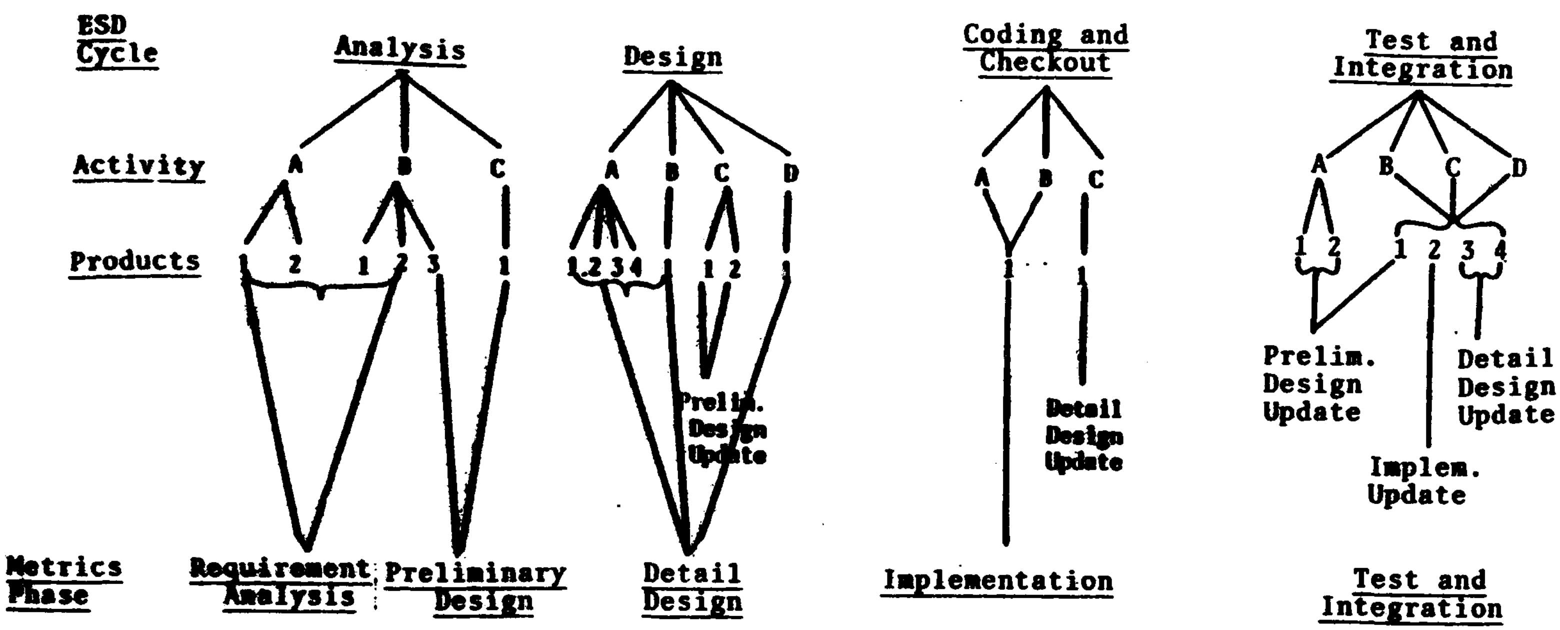


FIGURE II-2

MAPPING GUIDEBOOK COMPUTER PROGRAM
LIFE CYCLE MODEL INTO METRICS LIFE CYCLE MODEL
THROUGH ACTIVITIES AND PRODUCTS

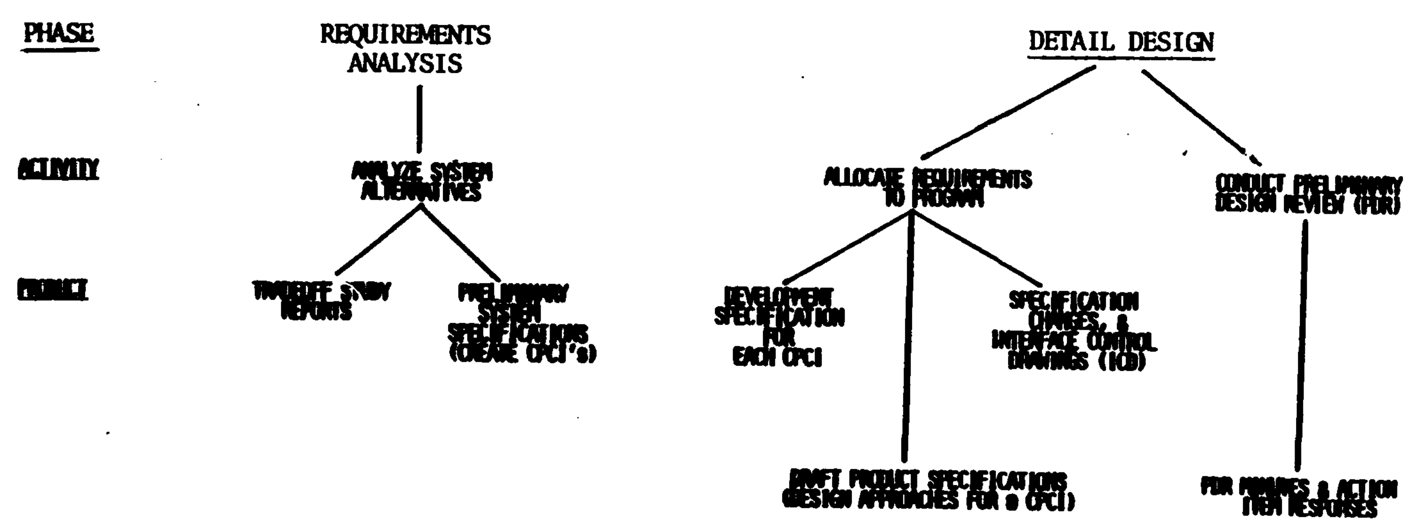


FIGURE II-3

METRICS LIFE CYCLE MODEL

MAPPED INTO ACTIVITIES AND PRODUCTS

In order to perform Detail Design phase Software Metrics, functional flowcharts, detailed flowcharts, data format descriptions, and descriptions of any additional algorithms are required. These products are generated while defining computer program logic and new algorithms, and when designing data storage structures. These products may be separate from, or included in, the preliminary Product specifications that are also required in the Detail Design phase. Preliminary Product specifications are generated when allocating computer program requirements internally. The final products required for the Detail Design phase are the minutes and action item responses generated during a Critical Design Review. FIGURE II-4 depicts the activities and products of the Detail Design phase. An update to the Preliminary Design Software Metrics is applied during this phase. This update process will be discussed in more detail in Section 2.4.2 "Update Information Requirements".

The information required to perform Implementation phase Software Metrics is the source code. This code is generated by the coding process itself, as well as through limited checkouts of compiler or assembly units. FIGURE II-5 details the activities and products required for the Implementation phase. An update to the Detail Design phase Software Metrics should be performed during this phase. The procedure will be discussed in more detail in Section 2.4.2 "Update Information Requirements".

2.4.2 Update Information Requirements

There are five update applications of Software Metrics performed during the software development life cycle. Preliminary Design metrics are reapplied twice, Detail Design metrics are reapplied twice, and Implementation metrics are reapplied once. TABLE II-B lists the update applications and the phases in which the updates are applied. "Steps for Update Applications" in the Module Instructions of each Quality Factor describes the detailed process for performing updates.

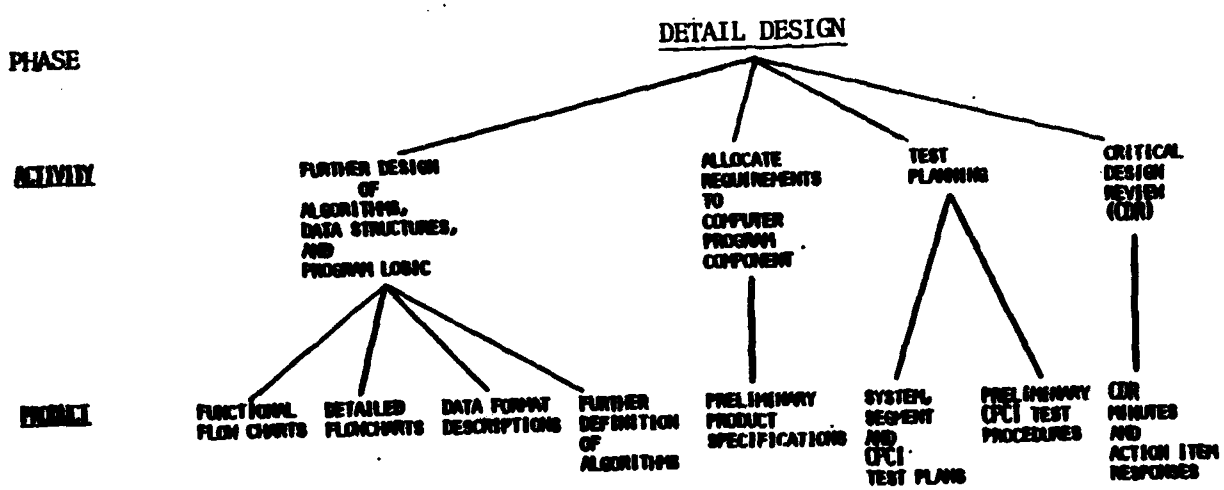


FIGURE II-4

METRICS LIFE CYCLE MODEL
MAPPED INTO ACTIVITIES AND PRODUCTS

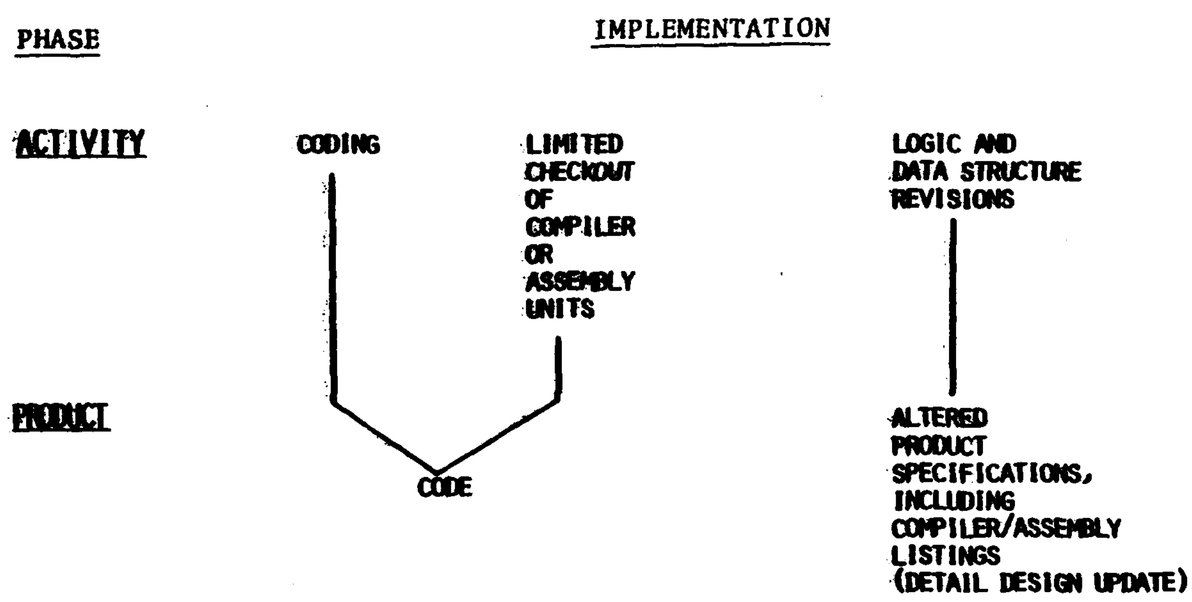


FIGURE II-5

METRICS LIFE CYCLE MODEL

MAPPED INTO ACTIVITIES AND PRODUCTS

UPDATE APPLICATION	PHASE WHEN UPDATES OCCUR
Preliminary Design	Detail Design
Detail Design	Implementation
Preliminary Design	Test and Integration
Implementation	Test and Integration
Detail Design	Test and Integration

TABLE II-B

UPDATE APPLICATIONS

Software Metrics is first re-applied to revised Preliminary Design documentation. Any segment Test Plans, System and CPCI Test Plans and preliminary CPCI Test Procedures are required for this update process. These products are generated during test planning activities in the Detail Design phase. FIGURE II-4 shows the Preliminary Design update products and activities.

The second re-application of Software Metrics is to revised *Detail Design documentation.* Altered Product Specifications, including compiler and/or assembly listings are required for this update application. These documents are produced during corresponding logic and data structure revision activities in the Implementation phase of the life cycle. FIGURE II-5 depicts the Detail Design update products and activities.

The final three update applications are re-applications of Software Metrics to Preliminary Design, Implementation and Detail Design documentation. The update to the Preliminary Design documentation requires final CPCI Test Procedures, any segment and system-level Test Procedures, and Test Reports. These products are generated during Test Planning activities, and module and CPCI tests and software subsystem integration activities in the Test and Integration phase. The update to the Implementation phase documentation requires computer program coding changes which are generated during module tests, CPCI tests, and software subsystem integration activities of the Test and Integration phase. The final update is applied to Detail Design phase documentation and requires modified Product specifications, and possible high-level specification and ICD changes. These documents are also generated during module and CPCI tests, and software subsystem integration activities in the Test and Integration phase. FIGURE II-6 details the activities and products of the Test and Integration phase.

2.4.3 Documentation Considerations

There are four major issues to consider when applying Software Metrics to the products generated during a project's life cycle.

(1) The content of the documents

(2) The possibility that the products may not map uniquely into discrete phases

(3) The appropriate level of application, System/Subsystem, or Subsystem/Module

(4) The sequencing of Metric applications.

2.4.3.1 Contents of Documents

The first issue to consider is the content of the documents. The type of information asked for by the Software Metrics should be found in the documents named in Subsection 2.2 and repeated in the Module Instructions. If these documents do not contain the necessary information, an investigation should be made into other contractor-supplied documentation to locate the sources of this information. If the information cannot be located, this may indicate a "gap" in the development and the issue should be resolved.

2.4.3.2 Product Mapping

The second issue involving documentation concerns the possibility that products may not always map

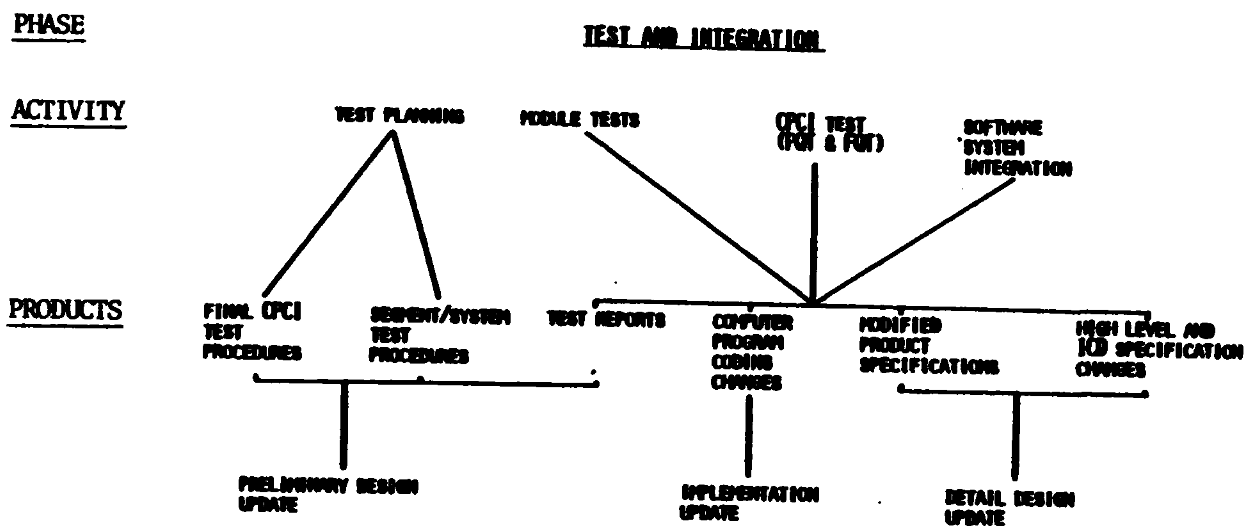


FIGURE II-6

METRIC LIFE CYCLE MODEL
MAPPED INTO ACTIVITIES AND PRODUCTS

uniquely into discrete life cycle phases. Thus, it may be necessary to refer to the documentation of previous phases to answer some of the Data Element questions in a particular phase.

2.4.3.3 Appropriate Levels of Applying Metrics

Software Metrics may be applied on two separate sets of levels; (1) System/Subsystem levels or (2) Subsystem/Module levels.

The third issue that concerns application of the metrics involves choosing the appropriate levels of application of the metrics. The metric system has two levels and can be applied either at the System/Subsystem levels or at the Subsystem/Module levels. If the decision is made to apply the metrics at the System/Subsystem levels one Handbook is required. For each Subsystem/Module an additional Handbook is required for the particular Subsystem/Module application.

A "System" is considered to be a collection of programs and/or subsystems sufficient to accomplish a significant, coherent application or function. Therefore, a "system-level" application of Software Metrics entails:

(1) Selecting Quality Factors for the entire system

(2) Applying Software Metrics to the documents that describe the collection of programs and/or subsystems that accomplish the mission of the system in order to determine the quality of those factors in the system.

For instance, before responding to the Data Element question:

"Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated?", as in Data Element "Unambiguous References", 2.4.3.3 continued

a system-level measurement in the Requirement Analysis phase requires searching the System specifications to determine if all of the subsystem's functions, their inputs and their outputs, have been clearly delineated. At the Preliminary Design phase, the Development specifications for all the subsystems should be searched to determine if all of the modules' functions, inputs and outputs for all of the subsystems have been clearly delineated before answering the same question. Throughout the metric application and life cycle the appropriate level System/Subsystem must be documented. Space for this is provided in the Handbook.

A "Subsystem" is a collection of modules organized in such a way that they accomplish a larger and more complex function than would normally be possible for a single program. Thus, a Subsystem/Module application of Software Metrics to each Subsystem entails:

(1) Selection of Quality Factors for each subsystem (may be a subset of those selected for the entire system)

(2) Applying Software Metrics to the documents that describe the collection of modules that accomplish the function of that subsystem in order to determine the quality of the selected factors for each subsystem.

In other words the Subsystem/Module Metrics application VIEWS the Subsystem as the "entire" system and the procedures over the entire life cycle are the same as in the System/Subsystem application. It is possible to apply the Metrics at both the System/Subsystem levels and the Subsystem/Module levels. This would require a separate Handbook and separate application.

Figure II-7 illustrates the two sets of possible applications where a system has three levels: (1) System, (2) Subsystem, and (3) Module.

2.4.3.4 Sequencing of Metric Application

The last issue to consider when applying the Software Metrics is the sequencing of the application effort itself. A decision should be made (in conjunction with the level of application) regarding desirable application sequences: should all applicable Software Metrics be applied to each document in turn, or should the Software Metrics be applied to all documents of a particular phase, searching each document for the response for each Data Element question?

2.5 APPLICABILITY OF SOFTWARE METRICS THROUGH THE LIFE CYCLE

As mentioned in Section I, "Software Metrics", each Quality Factor is calculated from Criteria, Metrics, and Data Elements. Most of the Quality Factors have at least one Data Element, Metric and Criteria for each of the four life cycle phases: Requirement Analysis, Preliminary Design, Detail Design, and Implementation. In addition, most of the Metrics within each Quality Factor are unique to a single phase. Therefore, for each Quality Factor, the normal progression throughout the life cycle is to apply the Software Metrics for each phase in the life cycle, asking different questions at each phase.

This normal progression is deviated from in two ways. First, some Quality Factors do not apply Software Metrics to all phases in the life cycle. Only the Quality Factors Reliability, Completeness, Efficiency, and Interoperability apply Software Metrics to all four phases. Integrity and Usability apply Software Metrics only to the Requirement Analysis and Preliminary Design phases, while Maintainability and Testability apply Software Metrics to all phases except the Requirement Analysis phase. Finally, Flexibility, Portability and Reusability apply Software Metrics only to the last two phases: Detail Design and Implementation. TABLE II-C shows this situation.

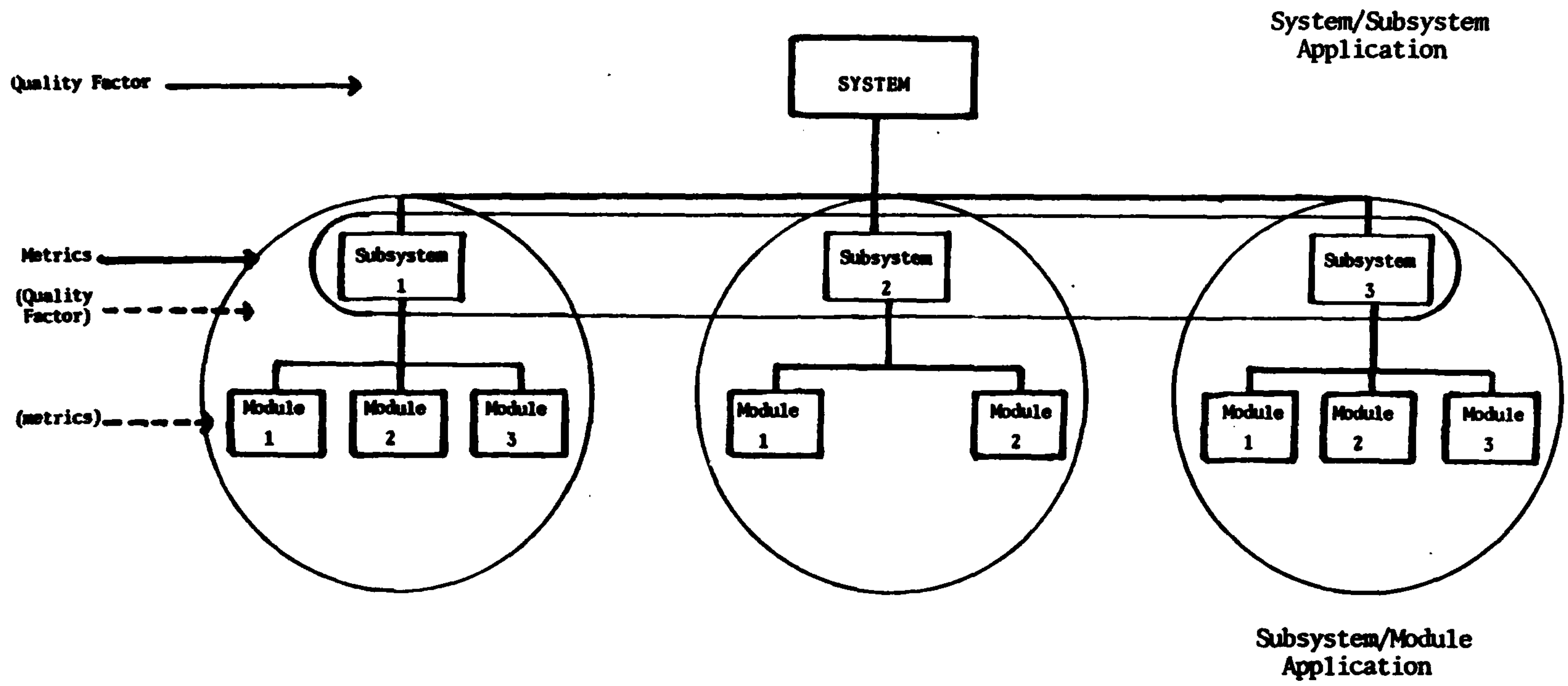


FIGURE II-7

LEVELS OF METRIC APPLICATION

QUALITY FACTOR	PHASES Where Applicable
Completeness	Requirements Analysis Preliminary Design Detail Design Implementation
Reliability	Requirements Analysis Preliminary Design Detail Design Implementation
Efficiency	Requirements Analysis Preliminary Design Detail Design Implementation
Integrity	Requirements Analysis Preliminary Design
Usability	Requirements Analysis Preliminary Design
Maintainability	Preliminary Design Detail Design Implementation
Flexibility	Detail Design Implementation
Testability	Preliminary Design Detail Design Implementation
Portability	Detail Design Implementation
Reusability	Detail Design Implementation
Interoperability	Requirements Analysis Preliminary Design Detail Design Implementation

TABLE II-C

QUALITY FACTORS ACROSS THE LIFE CYCLE

2.5 continued

The second deviation from the normal progression is that several Quality Factors repeat Metrics and Data Element questions across phases, or a Data Element name will be used, but a different (related) question will be asked. For instance, in the Quality Factor Correctness, the Metric "Completeness Checklist" is applied in the Requirement Analysis, Preliminary Design and the Detail Design phases. The same Data Element questions are asked in all three phases, but different documents and levels of detail are required in each phase. On the other hand, in the Metric "Data Consistency Measure", the Data Element "Global Data" asks the question "On the system level, is global data defined only once?" in the Preliminary Design phase, but asks, "On the module level, are global variables used as defined globally?" in the Detail Design phase. The difference between the two questions is the level of application. TABLE II-D outlines this situation.

Both of these deviations are a result of the nature of the Quality Factor being measured. The continuity of Software Metric's concepts is maintained.

DATA CONSISTENCY MEASURE	
Preliminary Design Phase	**Detail Design Phase**
1.0 Global Data (42) 1.1 On the system level, is global data defined only once? [Y/N] YES = 1, NO = Ø	1.0 Global Data (42) 1.1 On the module level, are global variables used as defined globally? [Y/N] YES = 1, NO = Ø

COMPLETENESS CHECKLIST	
1.0 Unambiguous References (2) 1.1 Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated? [Y/N] YES = 1, NO = Ø	1.0 Unambiguous References (2) 1.1 Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated? [Y/N] YES = 1, NO = Ø
2.0 External Data Reference (3) 2.1 Number of data references which are defined. 2.2 Number of major data references. Score = [2.1] ÷ [2.2]	2.0 External Data Reference (3) 2.1 Number of data references which are defined. 2.2 Number of major data references. Score = [2.1] ÷ [2.2]
3.0 Major Functions Used (4) 3.1 Number of defined functions used. 3.2 Number of functions identified. Score = [3.1] ÷ [3.2]	3.0 Major Functions Used (4) 3.1 Number of defined functions used. 3.2 Number of functions identified. Score = [3.1] ÷ [3.2]
4.0 Major Functions Defined (5) 4.1 Number of identified functions defined. 4.2 Number of functions identified. Score = [4.1] ÷ [4.2]	4.0 Major Functions Defined (5) 4.1 Number of identified functions defined. 4.2 Number of functions identified. Score = [4.1] ÷ [4.2]

TABLE II-D

DATA ELEMENTS ACROSS THE LIFE CYCLE

Preliminary Design Phase	Detail Design Phase
5.0 Decision Points Defined (6)	5.0 Decision Points Defined (6)
5.1 Is the flow of processing and all decision points in that flow defined? YES = 1, NO = ∅	5.1 Is the flow of processing and all decision points in that flow defined? YES = 1, NO = ∅
6.0 Agreement of Calling Sequence Parameters (7)	6.0 Agreement of Calling Sequence Parameters (7)
6.1 Number of defined and referenced calling sequence parameters that agree between functions.	6.1 Number of defined and referenced calling sequence parameters that agree between functions.
6.2 Number of calling sequence of parameters. Score = $\boxed{6.1} \div \boxed{6.2}$	6.2 Number of calling sequence of parameters. Score = $\boxed{6.1} \div \boxed{6.2}$
7.0 Problem Reports Resolved (8)	7.0 Problem Reports Resolved (8)
7.1 Number of those problem reports that have been closed (resolved)	7.1 Number of those problem reports that have been closed (resolved)
7.2 Number of problem reports related to the requirements that have been reported Score = $\boxed{7.1} \div \boxed{7.2}$	7.2 Number of problem reports related to the requirements that have been reported. Score = $\boxed{7.1} \div \boxed{7.2}$

TABLE II-D (cont'd)

DATA ELEMENTS ACROSS THE LIFE CYCLE

SECTION III

HANDBOOK FRAMEWORK

3.1 COMPONENTS OF COMPUTER SYSTEMS ACQUISITION METRICS HANDBOOK

The Computer Systems Acquisition Metrics Handbook consists of the following components:

General Instructions - This component discusses the Introduction to metrics, the Framework of Software Metrics, life cycle considerations, the framework of the Handbook, a step-by-step method on how to use the Handbook, and Quality Factor Selection. You are reading this component now.

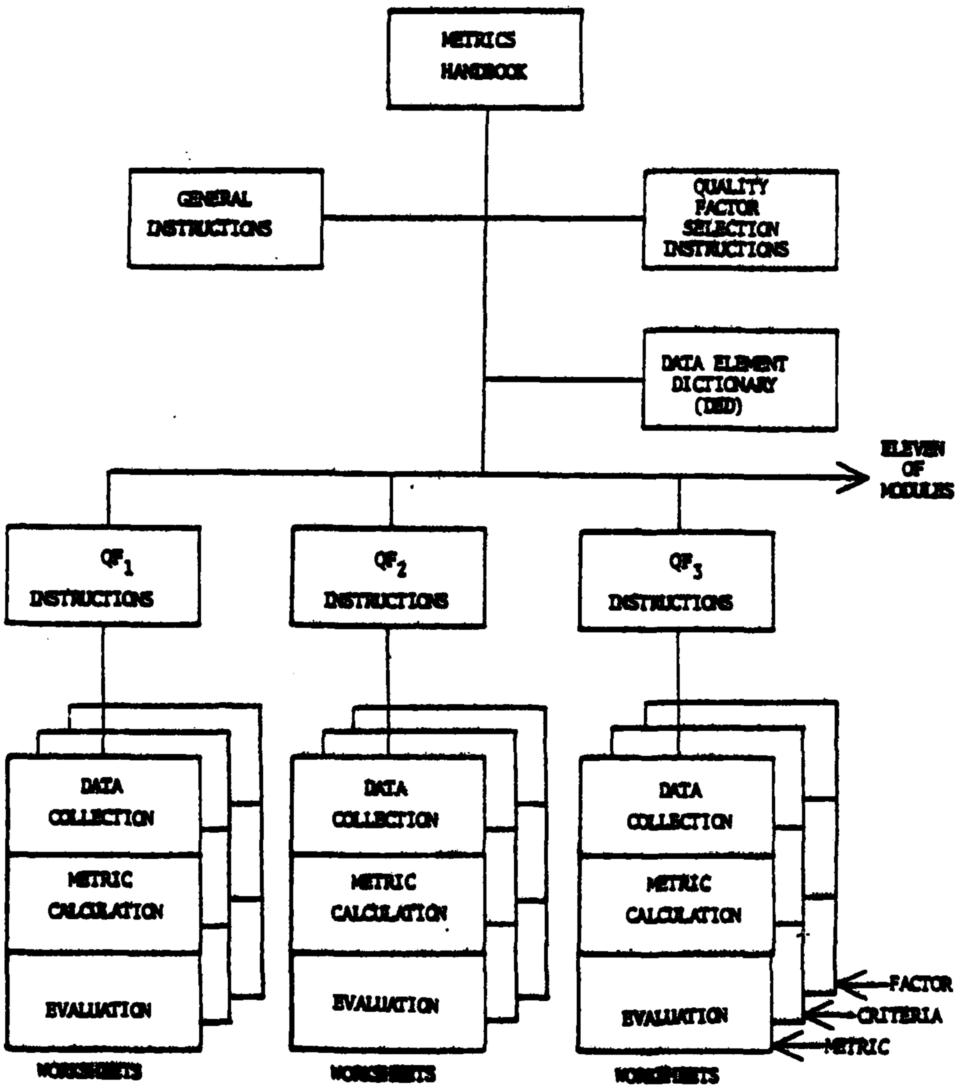
Eleven Quality Factor Modules - There are eleven quality factor modules, each module contains a complete hierarchy of worksheet sets with instructions on how and when to use them.

Data Element Dictionary (DED) - The DED is a reference guide for all the Data Elements. It lists the name, an index number for keying from the worksheet to the DED, Data Element questions asked, a life cycle phase description, an example with explanations and worksheet reference for each Data Element.

Figure III-1 graphically represents the framework of the Software Metrics Handbook. Section 3.2 discusses the relationship of the Handbook's framework to the Software Metric's framework.

3.2 INTERRELATIONSHIP OF SOFTWARE METRIC'S AND HANDBOOK'S FRAMEWORKS

The structure of the Handbook is designed to be closely connected with the Software Metrics framework. At the broadest level of detail where the two frameworks match is at the eleven modules themselves. Each module measures one Quality Factor.



HANDBOOK FRAMEWORK

FIGURE III-1

The worksheets in each module are arranged in two to four sets. These sets are organized according to the relevant Software Metrics Life Cycle Model's phases: Requirements Analysis, Preliminary Design, Detail Design, and Implementation. The worksheets within each phase are arranged accouding to their ranking in the Software Metrics framework from the bottom-up: (1) Data Elements, (2) Metric, (3) Criteria & (4) Quality Factor.. This allows them to be applied in a structured process. A worksheet set contains the following: one or more Metric worksheets, followed by one or more Criteria worksheets, followed by one Factor worksheet. Each level of worksheet contains scores from the next lowest level of measurement. Therefore, each Metric worksheet is composed of Data Element scores, each Criteria worksheet contains Metric scores, and each Factor worksheet is composed of Criteria scores. A person applying the Handbook at each phase utilizes the Handbook's worksheets in a process which is the exact inverse of the Software Metrics framework. The first value to be derived using the Software Metrics Handbook is the value of the lowest-level component in the Software Metrics hierarchy, the Data Element. The next value derived is the Metric value, which is one level higher, while the third value derived is the Criteria value which is the second-highest level in the hierarchy. The last value derived by applying the Handbook to a system or subsystem is the Factor score, which is the highest level component in the Software Metrics hierarchy. Using the modules' worksheets, the software Metrics framework is being completed starting at the bottom and working upwards. Figure III-2 depicts the interrelationship between the framework of the Software Metrics and that of the Handbook.

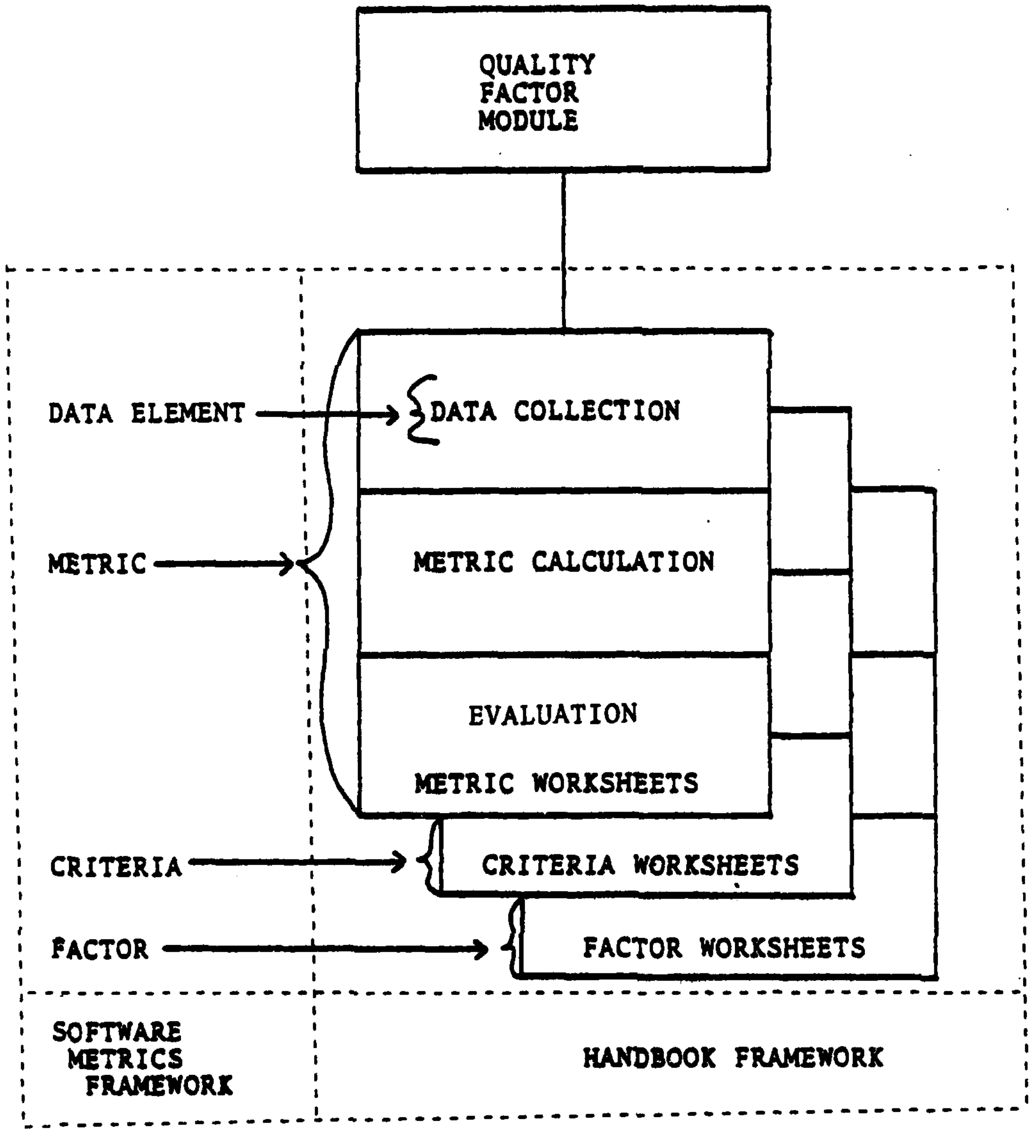


FIGURE III-2

INTERRELATIONSHIP OF HANDBOOK'S AND

SOFTWARE METRIC'S FRAMEWORKS

SECTION IV

HOW THE HANDBOOK WORKS

4.1 INTRODUCTION

This Handbook is an easy to use set of procedures to apply Software Metrics to different types of systems. Use of this Handbook provides a good working knowledge of Software Metrics and enables the user to apply them easily.

4.2 STEPS FOR USING THE SOFTWARE METRICS HANDBOOK

STEP 1 - SELECT QUALITY FACTORS

Section V of these General Instructions, "Quality Factor Selection", is the first step in applying Software Metrics. Quality Facotrs are selected either at System/Subsystem levels or Subsystem/Module levels or combinations of both sets of levels.

STEP 2 - OBTAIN RELEVANT MODULE

After selecting a particular set of Quality Factors for a system, the next step is to obtain the Quality Factor Modules corresponding to these Factors. There are eleven separate Quality Factor Modules, each one corresponding to a Quality Factor.

Each Quality Factor Module is composed of two parts: (1) Diagrams and instructions for completing the module's worksheets and score charts. Included in the instructions is a list of the activities and products required by each phase. And (2) Worksheet sets.

For a system/subsystem level application, only one module for each selected Quality Factor will be needed. If a subsystem/module application is desired, then the Quality Factors for that subsystem are selected and the corresponding modules are needed.

If a combination of System/Subsystem and Subsystem/Module applications are desired then Quality Factors are selected for the System and the corresponding modules are needed for application to the System Quality Factors are selected for each Subsystem and corresponding modules for each Subsystem are needed.

STEP 3 - COMPLETE WORKSHEETS

Worksheets are the tangible tools used in taking quantifiable measurements of software quality. The worksheets used in actual field work are contained in the Quality Factor Modules selected in STEP 2. The actual steps required for completing the worksheets can be found in "Steps for Completing Worksheets", in the Module Instructions for each Quality Factor. The end result of applying these worksheets will be the establishment of a quality rating for the Quality Factors.

STEP 4 - SUMMARIZING THE WORKSHEET RESULTS

After all of the worksheets have been completed for all of the selected Quality Factors at the end of each phase, the Score Charts are completed. The instructions for completing the Score Charts are included in "How to Use Score Charts sets", for each Quality Factor in the Module Instructions. The Score Charts provide a vehicle for summarizing the results of the Software Metrics Worksheet Applications.

4.3 IDENTIFICATION OF WORKSHEETS

In the upper right hand corner of each worksheet is the Form Code. Each worksheet is assigned a Form Code according to the Form Code Key notation in Table IV-A. When the worksheets are organized according to Form Code, the eleven "Quality Factor Modules" are formed.

MODULE PHASE LEVEL SEQUENCE

ItRAC.2

INTEGRITY REQ-ANA. CRITERIA SECOND PAGE OF SET

MODULE		
Co = Correctness	RA = REQUIREMENTS ANALYSIS	M = METRIC
Re = Reliability	PD = PRELIMINARY DESIGN	C = CRITERIA
Ef = Efficiency	DD = DETAIL DESIGN	F = FACTOR
It = Integrity	IM = IMPLEMENTATION	
US = Usability		
Ma = Maintainability		
Fx - Flexibility		
Te = Testability		
Po = Portability		
Ru = Reusability		
Ip = Interoperability		

TABLE IV - A

FORM CODE KEY

- The first segment of a Form Code is an abbreviation of the Quality Factor that pertains to the module that contains the worksheet: For example, CO (Correctness), Re (Reliability), etc.
- The second segment of a Form Code is an abbreviation of the phase of the set containing the worksheet: For example, RA (Requirements Analysis), PD (Preliminary Design), and IM (Implementation).
- The third segment of a Form Code is an abbreviation of the software quality measurement level that pertains to the worksheet: For example, M (Metric), C (Criterion), and F (Factor).
- The fourth segment of a Form Code is the sequence number of a particular worksheet (first Metric worksheet, second Metric worksheet, etc.).

4.4 DATA ELEMENT DICTIONARY (DED)

For quick reference, one hundred twenty-six (126) Data Elements are individually defined in the Data Element Dictionary portion of this Handbook. Each dictionary entry contains the following information: (1) Data Element name, (2) Index number, (3) Validation question associated with the Data Element in the Metric Worksheets, (4) Life-cycle phases(s) to which the Data Element pertains, (5) Definition, (6) An example of the Data Element's typical use, and (7) Explanation of how to score the Metric worksheets that contain the Data Element. EXHIBIT IV-1 is a typical entry of a Data Element in the Dictionary.

The Data Element Dictionary should be used as a reference when additional understanding of a Data Element question is desired, or when clarification is needed.

NAME: Share Temporary Storage

INDEX NUMBER: 91

DATA ELEMENT: Is temporary storage independent of other modules? /_Y_/_N_/

LIFE CYCLE PHASE(S): (1) Detail Design

DESCRIPTION: This is a binary measure to determine whether or not modules share temporary storage. It emphasizes the loss of module independence if temporary storage is shared between modules.

EXAMPLE: Storage should be separate for each module. Accessing commons should not be used instead of passing parameters from one routine to the next.

EXPLANATION: This is a binary measure answered by a "Yes" or a "No".

WORKSHEET REFERENCE:

FORM CODE	PAGE
MaDDM.5	Ma-28
FxDDM.3	Fx-18
TeDDM.3	Te-31
PoDDM.1	Po-16
RuDDM.1	Ru-16
IpDDM.1	Ip-30

EXHIBIT IV-1

EXAMPLE OF DATA ELEMENT FROM DATA ELEMENT DICTIONARY

SECTION V

QUALITY FACTOR SELECTION

5.1 INTRODUCTION

This section provides a discussion of a method for selecting the set of Quality Factors for a software system or subsystem. Following a determination of the appropriate Quality Factors for the system or subsystem, the corresponding Quality Factor Modules are applied to the system or subsystem.

The first step of the Quality Factor selection process is to examine the basic characteristics of the software system or subsystem, and then compile a preliminary list of Quality Factors that best provide for those characteristics. The eleven Quality Factors are listed and defined in TABLE I-A in Section I, "Software Metrics" of this Handbook.

A discussion of the applicability of Quality Factors to C^3 system is provided in Section 5.3 of these General Instructions. This preliminary list of Quality Factors can be developed by discussion with as wide a range as possible of the people responsible for the system development. For example, the Quality Factors input could come from the maintaining command, the originators of the requirements specifications or the potential users. The Quality Group responsible for measuring the quality of the system must obtain clean input on this selection to assure the appropriate Quality Factors are being measured. Some of the Quality Factors cannot co-exist in a system. Therefore the next step is a trade-off between conflicting Quality Factors to determine the final set of Quality Factors for measurement.

5.2 QUALITY FACTOR TRADE-OFFS

TABLE V-A shows either HIGH, LOW, or NEUTRAL relationships among the eleven Quality Factors. Because of the nature of the relationship between some of the Quality Factors, a high rank for some of the Quality Factors will result in a conflict with some

FACTORS	CORRECTNESS	RELIABILITY	EFFICIENCY	INTEGRITY	USABILITY	MAINTAINABILITY	TESTABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY	INTEROPERABILITY
CORRECTNESS											
RELIABILITY	○										
EFFICIENCY											
INTEGRITY			●								
USABILITY	○	○	●	○							
MAINTAINABILITY	○	○	●		○						
TESTABILITY	○	○	●		○	○					
FLEXIBILITY	○	○	●	●	○	○	○				
PORTABILITY			●			○	○				
REUSABILITY		●	●	●		○	○	○	○		
INTEROPERABILITY			●	●					○		

TABLE V-A

RELATIONSHIPS BETWEEN SOFTWARE QUALITY FACTORS

LEGEND: If a high degree of quality is present for Factor, what degree of quality is expected for the other:

○ = High ● = Low

Blank = No relationship or application dependent of the others. This is illustrated in TABLE V-A by a dark circle in the intersecting squares. In this situation a decision must determine the significance of the conflict between the Quality Factors. If the conflict is significant, one of the Quality Factors may be deleted from the list. However, if other considerations negate the impact of the conflict, both Quality Factors may be kept; however the low relationship between these factors must be kept in mind during Evaluation. TABLE V-B is a list and explanation of every possible Quality Factor conflict.

On the other hand, attaining high rank scores for certain Quality Factors may imply other Quality Factors will also be present. If the square in TABLE V-A contains a clear circle, a high degree of relationship exists between the factors. If the square is blank, a high degree of quality for one factor will have little or no effect on the other.

5.3 SIGNIFICANCE OF QUALITY FACTORS IN C^3 SYSTEMS

Since the main focus of this handbook is with Tactical Command, Control and Communication (C^3) Systems, the following paragraphs describe how applicable and important each Quality Factor is to a tactical environment based on typical characteristics of C^3 systems. The Quality Factors, with explanations of their significance, are listed in order of their importance.

- Correctness, Reliability and Efficiency - It is essential for a software system operating in a simulated or actual battle environment to perform quickly and accurately. That is the reason why the ability of a tactical C^3 system to satisfy its specifications, fulfill the user mission objectives, and perform its intended function with the least amount of resources and code are the most crucial requirements.
- Interoperability - An important goal of Air Force system developers is the development of a fully distributed TAF/C^3 system. This system, composed of many subsystems spread out geographically, will be designed to operate

INTEGRITY VS EFFICIENCY	The additional code and processing required to control the access of the software or data usually lengthens run time and require additional storage.
USABILITY VS EFFICIENCY	The additional code and processing required to ease an operator's tasks or provide more usable output usually lengthen run time and require additional storage.
MAINTAINABILITY VS EFFICIENCY	Optimized code, incorporating intricate coding techniques and direct code, always provides problems to the maintainer. Using modularity, instrumentation, and well commented high level code to increase the maintainability of a system usually increases the overhead resulting in less efficient operation.
TESTABILITY VS EFFICIENCY	The above discussion applies to testing.
PORTABILITY VS EFFICIENCY	The use of direct code or optimized system software or utilities decreases the portability of the system.
FLEXIBILITY VS EFFICIENCY	The generality required for a flexible system increases overhead and decreases the efficiency of the system.
REUSABILITY VS EFFICIENCY	The above discussion applies to reusability.
INTEROPERABILITY VS EFFICIENCY	Again the added overhead for conversion from standard data representations, and the use of interface routines decreases the operating efficiency of the system.
FLEXIBILITY VS INTEGRITY	Flexibility requires very general and flexible data structures. This increases the data security problem.
REUSABILITY VS INTEGRITY	As in the above discussion, the generality required by reusable software provides severe protection problems.
INTEROPERABILITY VS INTEGRITY	Coupled systems allow for more avenues of access and different users who can access the system. The potential for accidental access of sensitive data is increased as well as the opportunities for deliberate access. Often, coupled systems share data or software which compounds the security problems as well.
REUSABILITY VS RELIABILITY	The generality required by reusable software makes providing error tolerance and accuracy for all cases more difficult.

TABLE V-B

TYPICAL FACTOR TRADE-OFFS

using automated resource sharing and distributed control. For this reason, the ability of any one system to be coupled with another is a crucial factor.

- Flexibility and Maintainability - A tactical C^3 system will go through a complex evolution in its development, so it must be adaptable to constant change. Software modules in a system should be easy to add, replace, and change.

- Testability - A tactical C^3 system must go through extensive testing at different points of its development. Only after testing, can a system's reliability and performance in a battle environment be measured realistically.

- Portability - The ability to transfer data bases and/or software to and from any unit is important. A fully distributed C^3 system is composed of a configuration of mobile and non-mobile hardware units situated at different strategic locations.

- Usability - Another requirement crucial to the success of a tactical C^3 system in the field is that military personnel learn to utilize and interface with the software with ease. Some methods used by the Air Force to increase Usability are automated decision aids (to allow rapid information assimilation and decision making) and automated communication (to make information readily available to the commander and his staff).

- Integrity - The extent to which access to software and data can be controlled is relatively unimportant compared with the other requirements, but is essential nonetheless. For example, the interception of intelligence data by the enemy in a wartime situation could have grave implications.

- Reusability - The extent to which a program can be used in other applications is relatively unimportant in C^3 systems.

The order of importance is not always the same and the above order is meant only as a guide.

SECTION VI

QUALITY FACTOR EVALUATION

6.1 POST DATA COLLECTION

At the close of Section V you were instructed to use the Module Instructions to find directions for completing all worksheets. This subsection and post-data collection is included in the "Introduction and General Instructions" to discuss the possible uses of the data and metric values leading to Quality Factor Evaluations.

6.2 EVALUATION

As you will see when you start working with the modules each worksheet contains an Evaluation Worsection. It is important to remember that the worksection is a subjective worksection for recording the judgements of the person applying the module. The other worksections are part of the "objective" metric system and the "subjective" worksection is included for the purpose of future analysis. The Evaluation Worksection asks: "What is your evaluation of the reviewed products based on the Metrics above? _________(1-1Ø or Ø if you are unable to evaluate)." When a person is evaluating a Criteria it would be possible to have one Metric with a high score and another with a low score. The evaluator could decide that the Criteria should be evaluated fairly high (5-8) because in his opinion the low scoring Metric did not have much of an impact on the system. Because of this subjective nature, a particular evaluator may consistantly evaluate high, or consistantly evaluate low. Some method of tracking and monitoring this type of scoring should be developed so that an analysis of the scores and scorer can be done as a means of giving a proper interpretation to the historical data in the data base.

There are three dimensions that need to be analyzed over time to improve the credibility of threshold values leading to Quality Factor Evaluations: (1) The objective metric scores, (2) The subjective evaluation section score and (3) The independent evaluation of total systems that are in the maintenace phase of the life cycle. By comparing these three dimensions over time metric scores that consistantly have a strong or high correlation with evaluation at either the worksheet or system level will lead to the establishment of credible threshold value. When this happens the user for metrics will expand and become even more valuable.

ND

ATE